

# Parameter Passing Styles

Mattox Beckman

`beckman@iit.edu`

Illinois Institute of Technology

The function call is one of the most fundamental elements of programming. The meaning of a function call is greatly affected by the choice of parameter passing style.

- Understand five kinds of parameter passing:
  1. Call By Value
  2. Call By Reference
  3. Call By Name
  4. Call By Result
  5. Call By Value-Result
- Know how to implement each of these.

We will use the following code to illustrate the concepts:

```
1 let foo x y z =  
2   x := z * z * y;   (* let's pretend that this *)  
3   y := 5;           (* is legal *)  
4   x + y  
5  
6 let main () =  
7   let a = 10 in  
8   let b = 20 in  
9   foo a b (a+b)
```

- Parameters are evaluated before the function call takes place.
- The function receives a copy of the parameters.
  - Changes made to variables in the function are not visible outside.
- Advantages: speed
- Disadvantage: instability

```
1 # let pil a b = a;;  
2 val pil : 'a -> 'b -> 'a = <fun>  
3 # let rec foo () = pil 5 (foo ());;  
4 val foo : unit -> int = <fun>  
5 # foo ();;  
6 Stack overflow during evaluation (looping recursi
```

## Result of CBV

## §1 Parameter Passing Styles

```
1 let foo x y z =  
2   x := z * z * y;  
3   y := 5;  
4   x + y  
5  
6 let main () =  
7   let a = 10 in  
8   let b = 20 in  
9     foo a b (a+b)
```

- a is copied into x.
- b is copied into y.
- a+b is evaluated to 30, the 30 is copied into z.
- x is assigned  $30 * 30 * 20$ .
- y is assigned 5.
- upon return, a and b have their original values.

- This is used by C, C++, OCaml, ... “most languages”.

- Parameters are evaluated before the function call takes place.
- The function receives a copy of the parameters.
- Variables are passed as pointers.
  - Changes made to variables in the function are visible outside.
- Advantages: speed, saves some memory, side effects are possible when you want them.
- Disadvantage: side effects are possible when you don't want them.

## Result of Call by Reference

```
1 let foo x y z =  
2   x := z * z * y;  
3   y := 5;  
4   x + y  
5  
6 let main () =  
7   let a = 10 in  
8   let b = 20 in  
9   foo a b (a+b)
```

- a and x share the same memory.
- b and y share the same memory.
- a+b is evaluated to 30, the 30 is copied into z.
- x and a are assigned  $30 * 30 * 20$ .
- y and b are assigned 5.
- upon return, a and b have new values.

- Used by C, C++, OCaml optionally; Java by default.

**Example**

```
1 int inc(int i) {  
2     return ++i;  
3 }  
4  
5 int main() {  
6     int i = 10;  
7     cout << inc(i) << " " << i << endl;  
8 }
```

What will be the output of this code?



## Example

## §1 Parameter Passing Styles

---

```
1 int inc(int &i) {  
2     return ++i;  
3 }  
4  
5 int main() {  
6     int i = 10;  
7     cout << inc(i) << " " << i << endl;  
8 }
```

What will be the output of this code?

- Parameters are updated before the function call *returns*.
- Often combined with call by value. Call by result, call by value, and call by value-result are “subclasses” of call-by-copy. What changes is when the copy occurs.
  - Changes made to variables in the function are visible outside—in fact, that’s the whole point.
- Advantages: you can return multiple values from a single function
- Disadvantages: variables can be clobbered inadvertently.

## Result of Call By Result

```
1 let a = 10
2 let b = 20
3
4 let foo x y z =
5     x := z * z * y;
6     y := 5;
7     a + b
8
9 let main () =
10     foo a b (a+b)
```

- a is copied into x.
- b is copied into y.
- a+b is evaluated to 30, the 30 is copied into z.
- x is assigned  $30 * 30 * 20$ .
- y is assigned 5.
- a + b will evaluate 30
- upon return, x is copied into a, and y is copied into b.

- This is used by Prolog.

- Parameters are evaluated after the function call is made.
- The parameters are substituted into the function body.
- Changes made to variables in the function *are* visible outside.
- Advantages: stability
- Disadvantage: inefficiency — computations can be duplicated

```
1 # let pil a b = a;;  
2 val pil : 'a -> 'b -> 'a = <fun>  
3 # let rec foo () = pil 5 (foo ());;  
4 val foo : unit -> int = <fun>  
5 # foo ();;  
6 vat - : int = 5
```

## Result of Call By Name

```
1 let foo x y z =  
2   x * x + y * y  
3  
4 let main () =  
5   foo (10+10) (20+20)  
6   (main ())
```

- x is replaced by (10+10).
- y is replaced by (20+20).
- z is replaced by (main ()).
- The call to main via z never happens.
- The + operation happens four times.

- This was used by Algol. Also used by some “term rewriting” systems.

- Parameters are encapsulated into a *thunk*.
- The thunks are passed into the function.
- The first time a thunk is executed, the value is cached.
- Remaining executions use the cached value.
- Advantages: stability
- Disadvantage: efficient, but time sensitive.

```
1 # let pil a b = a;;  
2 val pil : 'a -> 'b -> 'a = <fun>  
3 # let rec foo () = pil 5 (foo ());;  
4 val foo : unit -> int = <fun>  
5 # foo ();;  
6 vat - : int = 5
```

## Result of Call By Need

```
1 let foo x y z =  
2   x * x + y * y  
3  
4 let main () =  
5   foo (10+10) (20+20)  
6   (main ())
```

- `x` is replaced by a pointer to `(10+10)`.
- `y` is replaced by a pointer to `(20+20)`.
- `z` is replaced by a pointer to `(main ())`.
- The call to `main` via `z` never happens.
- The `+` operation happens only once for each variable.

- This is used by Haskell. Also known as *lazy evaluation*.
- Not compatible with assignment.

(qdb 167) Consider the following code. What are the contents of `a`, `b`, `x`, `y` and function output after `foo` returns for each of the parameter passing styles?

```
1 let main () =  
2   let a = 5 in  
3   let b = 7 in  
4   let foo x y z =  
5     x := z + z + y;  
6     y := 9 + a;  
7     x + y  
8 in  
9   foo a b (a+b)
```



```

1 let main ( ) =
2   let a = 5 in
3   let b = 7 in
4   let foo x y z =
5     x := z + z + y;
6     y := 9 + a;
7     x + y
8 in
9   foo a b (a+b)

```

Style	a	b	x	y	Result
Value	5	7	31	14	45
V-Result	31	14	31	14	45
Ref	31	40	31	40	71
Name	31	40	n/a	n/a	71